University of Texas at Dallas

Department of Electrical Engineering

**EEDG 6304 – Computer Architecture**

Project #1 – Submission

Name : **Nadhiya Subramanian(2021187721)**

**Sharan Nagesh (2021166306)**

**Description of the analysis**:

Caches form the **integral part** of a microprocessor design, they are often called spinal cord for betterment of performance, cost, design wins etc. More research goes every year for an efficient cache design. In this project we take a simple step in analyzing the cache performance of an Alpha simple scalar machine with few design constraints.

Any cache design is proven to the industry/customers through their benchmarks. Right from **ARM, TI, Intel, microprocessors** go through strenuous benchmarking before bringing their products to the industry. Here in our example we are going to use **3 benchmarks** and analyze the cache design constraints.

The cache design parameters that can be tuned in our example are

* **Cache hierarchy** – One or two levels, for data and instruction caches
* **Unified caches** – Unified data and instruction caches for L1 and L2
* **Size** – Cache size is the most important factor to avoid capacity misses
* **Block size** – Usually taken around 64 bytes as optimal
* **Associativity** – Critical parameter to save conflict misses can be 1,2,4,8 way associative caches.

**One way is also known as direct mapped cache which obviously has more miss rate**.

While larger caches lure to bring better performance, they come along with higher cost too. So we are going to use the cost function to identify the optimal configuration.

CPI formulae used:  
  
 1. L1 separate and L2 separate  
  
CPI = CPI ideal + 5 \* ( L1InsMissRate \* %Ins + L1DataMissRate \* %Data) + 40 \* ( L2InsMissRate \* %Ins + L2DataMissRate \* %Data )  
  
2. L1 separate and L2 unified  
  
CPI = CPI ideal + 5 \* ( L1InsMissRate \* %Ins + L1DataMissRate \* %Data) + 40 \* ( L2MissRate \* %Data ) )  
  
3. L1 unified and L2 unified  
  
CPI = CPI ideal + 5 \* ( L1MissRate \* ( %Data) ) + 40 \* ( L2InsMissRate \* ( %Data ) ) )

# Part 2:

In this part, the CPI for the four individual benchmarks was calculated. Our baseline configuration will be the Alpha 21264 EV6 configuration:

- **Cache levels**: Two levels.

- **Unified caches**: Separate L1 data and instruction cache, unified L2 cache.

- **Size**: 64K Separate L1 data and instruction caches, 1MB unified L2 cache.

- **Associativity**: Two-way set-associative L1 caches, Direct-mapped L2 cache.

- **Block size**: 64 bytes.

- **Block replacement policy**: FIFO.

***GCC Benchmark :***

CPI = CPI ideal + 5 \* ( L1InsMissRate \* %Ins + L1DataMissRate \* %Data) + 40 \* ( L2MissRate \* ( L1InsMissRate \* %Ins + L1DataMissRate \* %Data ) )

Total number of Instructions = 337326966

Number of L1Ins access = 337326966

Number of L1Data access = 124102754

Number of L2 access = 3330162

L1 Ins miss rate = 0.0047, L1 Data miss rate = 0.0106, L2 miss rate = 0.1310

CPI = 1+5\*{(0.0047\*(337326966/337326966))+(0.0105\*(124102754/337326966))}+40\*{0.1310\*(124102754/337326966))]}

= **1.094**

## *Anagram Benchmark:*

Number of Instructions = 25593183

Number of L1Ins access = 25593183

Number of L1Data access = 11153900

Number of L2 access = 92638

L1 Ins miss rate = 0, L1 Data miss rate = 0.0049, L2 miss rate = 0.3189

CPI = 1+5\*{(0\*1)+(.0049\*(11153900/25593183))}+40\*{0.3189\*(11153900/25593183)]}

= **1.0568**

## *GO Benchmark:*

Number of Instructions = 545823529

Number of L1Ins access = 545823529

Number of L1Data access = 213791066

Number of L2 access = 1021473

L1 Ins miss rate = 0.0013, L1 Data miss rate = 0.0010, L2 miss rate = 0.0907

CPI = 1+5\*{(0.0013\*1)+(0.0010\*(213791066/545823529))}+40\*{0.0907\*(213791066/545823529))]}

= **1.015**

# Part 3:

Given a two-level cache hierarchy, 128KB available for L1 cache and 1MB available for L2 cache, identify the optimal configuration (in terms of achieved CPI) for each benchmark. Decision must be made between unified/separate caches, associativity, replacement policy etc.

The following design choice includes different associativity, replacement policies, and unified/separate caches for L1 and L2.

## Assumptions:

1. Both L1 and L2 use the same replacement policy. Example : if L1 cache uses FIFO, L2 cache also uses FIFO replacement policy.
2. Directly mapped design is also taken into consideration even though it gives poor performance, but it will help in analyzing the design better.
3. Associativity values range from 1, 2, 4 and 8. Design does not consider associativity more than 8 because they have much higher cost for a very little performance improvement in reality
4. Block size is just considered to be 64 Bytes. Having higher block size would decrease the number of lines in the cache and increases the miss penalty

So the number of iterations used for analysis for each benchmark namely, GCC, Anagram and Go

= three Cache combinations\*Block size \*Replacement Policy\*L1 set associativity\*L2 set associativity

= 3(L1 separate, L2 separate or L1 separate, L2 unified or L1 unified, L2 unified)\*1( 64)\*3(LRU, FIFO and Replacement)\*4(1 or 2 or 4 or 8)\*4(1 or 2 or 4 or 8)

= 3\*1\*3\*4\*4

No of iterations =144 for each benchmark

The below graphs show the CPI plotted against various configuration for L1 Separate-L2 separate, L1 Separate L2 unified, and L1,L2 unified for all three benchmarks

**GCC Benchmark**:

**Anagram Benchmark:**

**GO Benchmark:**

**Conclusion for part 3**:

The graphs plotted above gives us a clear picture for the design choices

Optimum CPI in each case:



**GO Benchmark**:

L1 unified 64 block size, 8 way set associative and Random replacement Policy.

L2 unified 64 block size, 4 way set associative and Random replacement Policy.

CPI Optimum = 1.0092

**ANAGRAM Benchmark**:

L1 separate 64 block size, 2-way set associative and LRU replacement policy

L2 unified 64 block size, 1-way set associative and LRU replacement policy

CPI Optimum = 1.055

**GCC Benchmark**:

L1 Separate 64 block size, 8-way set associative and LRU replacement Policy

L2 unified 64 block size, 8-way set associative and LRU replacement Policy.

CPI Optimum = 1.038

**Part 4 :**

**Cost Function :**

Let the

Associativity of L1 be A1

Associativity of L2 be A2

Replacement policy be R

Block Size be BS

Cost of Splitting L1 be S1

Cost of splitting L2 be S2

Assumption:

Let R for Random = 0

R for FIFO = 1.03

R for LRU = 1.0815

S1 = 2

S2 = 1

Base Cost = 100 units

When the associativity of the cache increases the area of the cache also increases, which results in increase in cost of the cache. So, cost is proportional to the associativity. Hence, we have

Cost = (A1\* Base cost) + (A2\* base cost)

Since the L1 cache is of smaller size with greater associativity and is placed near the processor than L2 cache, L1 cache is faster than L2 cache. Assuming the L1 cache is 1.2 times costlier than cost of L2 cache.

Cost = (A1+(A2/1.2))\*Base cost

Considering the replacement policy, changing form Random to FIFO increases the cost only by a smaller amount. LRU replacement is much costlier than random and FIFO. Assuming that changing the replacement for Random to FIFO increases the cost only by 3% and changing the replacement for Random to LRU increases the cost by 8% . So, changing the replacement policy form FIFO to LRU increases the cost by 5%. Hence, we got the above values for replacement R.

Splitting the cache also results in increase in cache cost. Since they require additional hardware and it also results in increased bandwidth. Hence the cost for splitting the cache is also added to the total cost.

Hence, Total cost = ((A1+(A2/1.2)+R)\* Base cost)+ S1+S2

**Part 5:**

**Optimize cache for performance/cost:**

To find the optimal configuration, the cache configuration such as associativity, Replacement policy and cache type is considered along with the cost. A plot of Cache configuration for each benchmark is plotted against the product of the CPI and cost. The configuration which gives the lowest value among all in the graph is considered as the optimal configuration.

**GCC Benchmark:**

**Anagram Benchmark:**

**GO Benchmark:**

Hence, from the graph above the optimum configuration would be,

GCC : L1 separate L2 unified with CPI = 1.0422 , replacement policy = LRU , Cost = 676.816, A1=4 and A2 = 2

Anagram : L1 separate L2 unified with CPI = 1.055 , replacement policy = FIFO , Cost = 588.33, A1=4 and A2 =1

Go : L1 separate L2 unified with CPI = 1.0099 , replacement policy = LRU , Cost = 593.48, A1=4 and A2=2

Hence, the optimum configuration for all three benchmark together would be L1 separate and L2 unified , Replacement policy LRU with associativity as 4 for L1 and 2 for L2.

**Conclusion:**

Thus, the best possible configuration and its cost function have been computed by modifying various parameters such as cache type, associativity and replacement policies.

**Script for benchmarks:**

#!/usr/bin/perl

## Script to run the cache benchmarks and analyze the results

use warnings;

use POSIX;

use Getopt::Long;

our $options =();

GetOptions(

"benchmark=s" => \$options{benchmark},

);

my @assoc\_arr = ("1", "2", "4", "8");

my @cache\_type = ("unified", "Sep\_L1\_only","SepL1L2");

#r- random, f-fifo, l - lru

my @repl\_policy = ("r", "f", "l");

my @L1d\_sep\_size = ("65536");

my @L2d\_sep\_size = ("524288");

## Initial cache size L1 = 128KB, L2 = 1MB, Block Size= 64Bytes

my $L1\_cache\_size = 128\*1024;

my $L2\_cache\_size = 1024\*1024;

my $blk\_size = 64;

my $L1d\_sets;

my $L1i\_sets;

my $L2d\_sets;

my $L2i\_sets;

my $L1d\_ass\_way;

my $L1i\_ass\_way;

my $L2d\_ass\_way;

my $L2i\_ass\_way;

my $L1u\_sets;

my $L2u\_sets;

my $L1i\_sep\_size;

my $run\_cmd = "";

# Loop conditions

foreach my $cache\_type (@cache\_type) {

if($cache\_type eq "unified") {

foreach my $repl\_policy (@repl\_policy) {

foreach my $L1\_way (@assoc\_arr) {

$L1u\_sets = ceil($L1\_cache\_size/($blk\_size \* $L1\_way));

foreach my $L2\_way (@assoc\_arr) {

#Unified cache so consider only d and set the same for I

$L2u\_sets = ceil($L2\_cache\_size/($blk\_size \* $L2\_way));

$run\_cmd = "./sim-cache -cache:il1 dl1 -cache:dl1 ul1:$L1u\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 none -cache:dl2 ul2:$L2u\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/cc1.alpha -O ./benchmarks/1stmt.i | tee GCC\_L1L2Unified\_outputs/gcc\_unified\_$repl\_policy-$L1u\_sets-$L2u\_sets.txt";

print "GCC Executing the cmd $run\_cmd \n";

system($run\_cmd);

$run\_cmd = "./sim-cache -cache:il1 dl1 -cache:dl1 ul1:$L1u\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 none -cache:dl2 ul2:$L2u\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/anagram.alpha ./benchmarks/words \<./benchmarks/anagram.in\> OUT | tee Anagram\_L1L2Unified\_outputs/anagram\_unified\_$repl\_policy-$L1u\_sets-$L2u\_sets.txt";

#print "Executing the cmd $run\_cmd \n";

#system($run\_cmd);

$run\_cmd = "./sim-cache -cache:il1 dl1 -cache:dl1 ul1:$L1u\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 none -cache:dl2 ul2:$L2u\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/go.alpha 50 9 ./benchmarks/2stone9.in | tee GO\_L1L2Unified\_outputs/go\_unified\_$repl\_policy-$L1u\_sets-$L2u\_sets.txt";

print "Executing the cmd $run\_cmd \n";

system($run\_cmd);

} #L2\_assoc loop close

} # L1\_assoc loop close

} # repl\_policy loop close

} #unified if loop close

elsif ( $cache\_type eq "Sep\_L1\_only") {

foreach my $L1d\_sep\_size (@L1d\_sep\_size) {

$L1i\_sep\_size = (128\*1024)-$L1d\_sep\_size;

foreach my $repl\_policy (@repl\_policy) {

foreach my $L1\_way (@assoc\_arr) {

$L1d\_sets = ceil($L1d\_sep\_size/($blk\_size \* $L1\_way));

$L1i\_sets = ceil($L1i\_sep\_size/($blk\_size \* $L1\_way));

foreach my $L2\_way (@assoc\_arr) {

#Unified cache so consider only d and set the same for I

$L2u\_sets = ceil($L2\_cache\_size/($blk\_size \* $L2\_way));

$run\_cmd = "./sim-cache -cache:il1 il1:$L1i\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:dl1 dl1:$L1d\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 none -cache:dl2 ul2:$L2u\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/cc1.alpha -O ./benchmarks/1stmt.i | tee GCCSepL1\_outputs/gcc\_sepL1\_$repl\_policy-$L1d\_sets-$L2u\_sets.txt";

print "Executing the cmd $run\_cmd \n";

system($run\_cmd);

$run\_cmd = "./sim-cache -cache:il1 il1:$L1i\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:dl1 dl1:$L1d\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 none -cache:dl2 ul2:$L2u\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/anagram.alpha ./benchmarks/words \<./benchmarks/anagram.in\> OUT | tee AnagramSepL1\_outputs/anagram\_sepL1\_$repl\_policy-$L1d\_sets-$L2u\_sets.txt";

#print "Executing the cmd $run\_cmd \n";

#system($run\_cmd);

$run\_cmd = "./sim-cache -cache:il1 il1:$L1i\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:dl1 dl1:$L1d\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 none -cache:dl2 ul2:$L2u\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/go.alpha 50 9 ./benchmarks/2stone9.in | tee GoSepL1\_outputs/go\_sepL1\_$repl\_policy-$L1d\_sets-$L2u\_sets.txt";

print "Executing the cmd $run\_cmd \n";

system($run\_cmd);

} #L2\_assoc loop close

} # L1\_assoc loop close

} # repl\_policy loop close

} # L1d-sep\_size

} # elsif loop close

else {

foreach my $L2d\_sep\_size (@L2d\_sep\_size) {

$L2i\_sep\_size = (1024\*1024)-$L2d\_sep\_size;

foreach my $L1d\_sep\_size (@L1d\_sep\_size) {

$L1i\_sep\_size = (128\*1024)-$L1d\_sep\_size;

foreach my $repl\_policy (@repl\_policy) {

foreach my $L1\_way (@assoc\_arr) {

$L1d\_sets = ceil($L1d\_sep\_size/($blk\_size \* $L1\_way));

$L1i\_sets = ceil($L1i\_sep\_size/($blk\_size \* $L1\_way));

foreach my $L2\_way (@assoc\_arr) {

#Unified cache so consider only d and set the same for I

$L2d\_sets = ceil($L2d\_sep\_size/($blk\_size \* $L2\_way));

$L2i\_sets = ceil($L2i\_sep\_size/($blk\_size \* $L2\_way));

$run\_cmd = "./sim-cache -cache:il1 il1:$L1i\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:dl1 dl1:$L1d\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 il2:$L2i\_sets:$blk\_size:$L2\_way:$repl\_policy -cache:dl2 dl2:$L2d\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/cc1.alpha -O ./benchmarks/1stmt.i | tee GccSepL1L2\_outputs/gcc\_sepL1L2\_$repl\_policy-$L1d\_sets-$L2d\_sets.txt";

print "Executing the cmd $run\_cmd \n";

system($run\_cmd);

$run\_cmd = "./sim-cache -cache:il1 il1:$L1i\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:dl1 dl1:$L1d\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 il2:$L2i\_sets:$blk\_size:$L2\_way:$repl\_policy -cache:dl2 dl2:$L2d\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/anagram.alpha ./benchmarks/words \<./benchmarks/anagram.in\> OUT | tee AnagramSepL1L2\_outputs/anagram\_sepL1L2\_$repl\_policy-$L1d\_sets-$L2d\_sets.txt";

#print "Executing the cmd $run\_cmd \n";

#system($run\_cmd);

$run\_cmd = "./sim-cache -cache:il1 il1:$L1i\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:dl1 dl1:$L1d\_sets:$blk\_size:$L1\_way:$repl\_policy -cache:il2 il2:$L2i\_sets:$blk\_size:$L2\_way:$repl\_policy -cache:dl2 dl2:$L2d\_sets:$blk\_size:$L2\_way:$repl\_policy -tlb:itlb none -tlb:dtlb none ./benchmarks/go.alpha 50 9 ./benchmarks/2stone9.in | tee GoSepL1L2\_outputs/go\_sepL1L2\_$repl\_policy-$L1d\_sets-$L2d\_sets.txt";

print "Executing the cmd $run\_cmd \n";

system($run\_cmd);

} #L2\_assoc loop close

} # L1\_assoc loop close

} # repl\_policy loop close

} # L1d-sep\_size

} # L2d-sep\_size

} #else loop close

} #main loop close